O'REILLY®

Compliments of PayPal

Understanding the InnerSource Checklist

How to Launch Collaboration Within Your Enterprise

Silona Bonewald

YOU DON'T HAVE TO GO THERE ALONE

Along the way you will face the forces of time pressures, competitors, budgets, old code, missing docs, obscure architectures, dead languages, intransigence, clueless managers, low morale, entrenched processes, power struggles, acquisitions, mergers, personal fiefdoms, time zones, ...

LEARN	USE	SHARE
Training	Tools	Vision
Studies	History	Culture
Patterns	Templates	Results

BECAUSE NONE OF US HAVE TIME TO MAKE ALL THE MISTAKES

INNERSOURCECOMMONS.ORG

9 781491 986929

Understanding the InnerSource Checklist

How to Launch Collaboration Within Your Enterprise

Silona Bonewald

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Understanding the InnerSource Checklist
by Silona Bonewald

Copyright © 2017 O'Reilly Media. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com/safari*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Andy Oram
Production Editor: Melanie Yarbrough
Copyeditor: Octal Publishing Services
Proofreader: Charles Roumeliotis
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

May 2017: First Edition

Revision History for the First Edition
2017-04-12: First Release

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Understanding the InnerSource Checklist*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-491-98692-9

[LSI]

Table of Contents

Foreword v

1. Why InnerSource? 1
Our Audience 3
What Does Open Source Have That I Don't Have? 3
Open Source Today 4
Open Source's Future in the Commercial World: InnerSource 4
A Brief History of InnerSource 5
What Lies Behind Open Source Practices 6

2. What InnerSource Is and Isn't 9
We Have GitHub Enterprise, So We Must Be InnerSource! 10

3. The Most Important Role, and the First Step: Trusted Committer 15
Defining the Role 16
Refining the Role 17
Immediate Benefits 18
Rewarding TCs 18

4. Passive Documentation and the Need for Findability 21
Creating Passive Documentation 21
"Did You Read the FINE Manual?" 22
Findability 23

5. Creating Good House Rules for Guests: Writing Contributing Agreements..... 25
What Is a Contributing Agreement? 26
Mi Casa Es Su Casa 27
Win/Win 27
One Size Fits All? 27

6. Working Within the Enterprise: Understanding Planning..... 29
Keep It Small and Simple, and Engage Your Staff 30
Planning and Product Specialists 31
Inclusion and Transparency 31
Planners Can Have an Impact on Processes 32
Results 33
Crossing the Gap from Planning to Developers 33

7. From Internal Silos to Internal Transparency..... 35
Where Did Silos Come From? 35
What's Wrong with Silos? 36
Transparency for Community Sourcing 36
Transparency Boosts Decision-Making 37
How Do We Break Down Silo Walls? 37
Findable Documentation Is Part of Transparency 38
Where Do We Still Need to Improve Transparency? 39
What Are the Limits or Pitfalls in Enterprise Transparency? 39

8. Looking Forward..... 41
Creating an Industry Standard 41

9. Appendix..... 43
The Actual Checklist 43

Foreword

PayPal first spoke about its InnerSource journey at OSCON North America in 2015. We didn't claim to have all the answers, just a will to experiment and openly report on our findings as we went about our journey to adopt open source methodologies within PayPal to reduce engineering silos and increase cross-stack collaboration.

A key part of our InnerSource journey has been building a team capable of designing tools and processes that can help us make this cultural shift. Silona Bonewald's experience with open source and with product management along with her love of data science made her the ideal person to implement InnerSource across all of PayPal.

Silona likes order. She makes checklists to ensure that she doesn't forget small details, but also to establish implementation norms and streamline adoption of new ideas. She's written this booklet to share some of the thinking that went into our InnerSource Implementation Checklist. We hope you enjoy it and benefit from it.

— Danese Cooper, head of
Open and InnerSource, PayPal

CHAPTER 1

Why InnerSource?

A group of us in the open source community feel strongly that we can make work better by introducing and adopting open source principles and processes to larger enterprises. This includes attributes that benefit the company (faster development, better cross-team collaboration, more documentation) and an ethos that benefits the workers (mentoring processes, accountability, and a supportive community).

It's a big goal. We started an organization called InnerSource Commons to share information and ideas among organizations working with InnerSource. We talk often about perfection being the enemy of action. That's one reason we focus on the smallest possible steps to effect change.

At the Commons, we also believe that when companies fundamentally understand many of the methods of open source, they can be confident and productive actors in the open source community. InnerSource is a way to bring them in while respecting their limits. InnerSource opens teams and departments within a company, but does not release proprietary information. It has been shown to be effective at reducing silos, increasing cross-stack understanding, and even stimulating innovation.

In good open source tradition, we are writing this book to share some of what we in the InnerSource community are doing to bring open source tools and methodologies to the enterprise environment. At the end, we present a checklist that quickly lays out the tasks that different parts of an organization have. It also lets you see how far

your organization has come in implementing an InnerSource project. Our implementation of InnerSource adds common-sense steps as a recommended path, with a goal of "InnerSource-Ready" certification for groups completing the steps.

The main point of this book is to find simple ways to encourage fundamental changes to typical corporate behavior.

One of the great things about InnerSource is that it doesn't need to begin—in fact, probably *should not* begin—as a top-down mandate from headquarters. Just one team in one department can make a few small changes in the right places to see results. And, hopefully, other teams will be inspired enough to follow.

Many InnerSource processes were born out of errors or problems. In fact, one of our biggest strengths is our ability to learn from errors—those we make, and those that others make and then share. Others will help us learn if we let them. That's one reason we encourage transparency.

For example, people working on product integration often find it easier to send a series of private email exchanges or hold meetings among a fraction of the people planning the integration than to bring all stakeholders into the process. Requiring all of those involved to collaborate transparently has enormous payoffs,[1] especially if you do it in a way that can be archived (in a discoverable location) so that other people can learn from it.

InnerSource is enabled by tools and processes, but it is also a change to the culture. The biggest change is allowing mistakes, talking about them, and learning from them.

This book is a true exercise within this premise. We are putting it out in the open, rough edges and all, explaining lessons we've learned along the way, and sharing the solutions we have found. It will be posted on the InnerSource Commons site, where people can comment on it and help it grow. We will print an official copy, of course. But because we strive to live in the "Pull Request Culture" we are creating, if you're reading the hardcopy and see anything wrong or feel the need to add more to the conversation, please contribute your feedback online.

1 More on this in Chapter 6.

We understand that it can be difficult in a business environment to share feedback freely when *faux pas* in brand management have financial repercussions. At the Commons, we work under *Chatham House Rules* (see the section "A Brief History of InnerSource" later in this chapter) so that people can feel confident that nobody is reporting on their involvement until they are ready to go public. Likewise, with this book we have changed some names to protect the innocent, so to speak.

We hope that as we go on this journey, you will see how taking advantage of small changes can begin to make larger cultural change a reality. And, yes, some serious change management techniques are proposed here.

Our Audience

We strive to include something for everyone in this book. Developers can learn what it's like to be either a contributor or a *Trusted Committer* (more on this role a bit later) who vets the contributions. Product owners and product specialists can make large gains through reuse, collaboration, and integrations. Planners will better understand how to manage the changes that InnerSource brings and will learn how helping teams negotiate the complexities of integration and collaboration can reduce tribal knowledge and technical debt. And, finally, upper management will find new ways to improve employee satisfaction and to integrate new business units and acquisitions.

What Does Open Source Have That I Don't Have?

Briefly, open source has flexibility, synergy across groups because of transparency, a culture that fosters collaboration, and a combination of standardization and easy-to-find documentation that greatly improves the learning curve. Open source has developers that participate due to intrinsic motivations, an ethos of honoring mentors, and a view of contributions as a gift, not a burden. Transparency and widespread contributions lead to software that better meets the needs of the users.

Open Source Today

Open source software has "won." Every *Fortune 500* company uses or works on some kind of open source project. Sonatype, a major player in the open source community, conducted a survey in 2014 of large enterprises and found that "more than 90 percent of a typical application is now open source components."[2] One major advantage of open source software is that it has consistently shown a lower defect density than the industry average.[3]

Open Source's Future in the Commercial World: InnerSource

But how do the strengths of open source help *within* a company? Realistically, most companies cannot be strictly open source, because regulatory and commercial requirements forbid them from sharing their source code. This is where InnerSource comes in. InnerSource is a method of applying lessons learned in the open source software movement to companies that are developing software internally.

InnerSource can help corporations become better actors in the open source community, while bringing the advantages of open source to the corporate world. Our most important goals are the following:

- To help the enterprise learn how to improve collaboration
- To help the enterprise create cleaner code
- To reduce bottlenecks
- To facilitate integrations between teams

In most enterprises, it is difficult to make significant changes quickly. Even when it's possible, rapid cultural or process change can be more disruptive than helpful. This goes double for when the changes are mandated from the top without buy-in from the people in the trenches. InnerSource works by starting with the smallest

2 Wayne Jackson, "The 2014 Survey: Marked by an Industry Shock Wave", The Nexus, June 20, 2014.

3 In Coverity's annual static code analysis reports, most recently, the "Coverity Scan Report".

steps possible to effect change, and by making meaningful compromises to adapt to circumstances. This minimizes disruption and gives people a chance to see how effective it is before making larger steps. In fact, just a single team in one department can effectively adopt InnerSource.

A Brief History of InnerSource

Deciding to apply open source methodologies on an enterprise level is neither new nor unique. Many people have worked on similar projects for almost as long as open source has existed. It is a natural decision because so many people enjoy working on open source projects and want to bring that ethos into their work environment. Many names have been used to describe this process of translating open source to the enterprise, from "internal open source," "enterprise open source," and "visual source" to "corporate open source," but few have succeeded for long. The term we are using was coined by Tim O'Reilly more than 15 years ago. Originally, it was "Inner Source," but we removed the space between the words so that the term is findable in a search.

InnerSource as a movement or method began with a conversation among a group of us in the open source community who were independently working to bring the open source ethos to the commercial world. We created a consortium in true open source fashion[4] to create and maintain InnerSource definitions and standards. This way, open source leaders are able to maintain the ethos and culture of the true meaning of InnerSource, even in the sometimes-difficult enterprise environment. One key element of this process has been our fervent adoption of *Chatham House Rule*:

> When a meeting, or part thereof, is held under the Chatham House Rule, participants are free to use the information received, but neither the identity nor the affiliation of the speaker(s), nor that of any other participant, may be revealed.

The simplicity of the Chatham House Rule embodies what we are working toward with InnerSource. Creating simple rules that are easy to follow gives us maximum leverage to effect change. Transparency in a commercial environment has been a huge hurdle. This

4 Specifically, Apache Software Foundation style.

rule addresses commercial enterprises' fear of collaboration with potential competitors and allows us to be more open with one another about what we are trying to do. It allows us to admit our failures and to share information and complaints with our peers so that we can work together as a group to quickly solve our problems.

The Chatham House Rules are a compromise that open source did not need to make to survive, but that has been crucial to InnerSource's creation and growth. It is pleasingly symbolic that we are using this tool of transparency and openness—along with a crucial dose of privacy—to create our new definition of InnerSource. It perfectly illustrates the dichotomy we are balancing.

What Lies Behind Open Source Practices

A long tradition of jokes and humorous stories show children or unsophisticated people acting out things that they've seen other people do—for instance, building a nonfunctional plane from bamboo in a cargo cult—without knowing why. The humor springs from the absurdity of actions taken out of the original context where they made sense. Unfortunately, too many practices adopted by businesses from open source projects fail for the same lack of understanding. What makes it even more difficult to adopt open source practices intelligently is that many open source practitioners talk about them enthusiastically without understanding why they worked in the open source setting.

Most business documents stress *how* to do things, but not *why*. Business environments often move too quickly to solidify processes. This report lays out the *why*s so that you set up the right environment in which your new practices are likely to succeed. We put the checklist for starting InnerSource at the end of the report because we first want to give context and human stories, showing you why we came up with the processes we did. Bringing InnerSource to enterprises is a complex undertaking, and what we did might not work for everyone else. We prefer to explain why we decided we need a new rule and *why* a process works for us so that you can create your process—your own *how*—to fit your needs.

Of course, you can skip to the checklist in this book or at this book's website if you're eager to move on. But we've made it easy to skim the *why*s: Each chapter begins with a TL;DR (Too Long; Didn't

Read) that sums up a problem we found, the smallest possible step to move toward a solution, and why that solution works.

CHAPTER 2

What InnerSource Is and Isn't

NOTE

TL;DR

Too often, enterprise culture wants to change the definition of InnerSource to something more familiar. We must help enterprise stakeholders create clear practices and definitions in order to maintain the culture of open source as much as possible while still highlighting the benefits to the enterprise.

One of the major problems we have encountered when implementing InnerSource has been, at its root, a vocabulary problem. After we completed several successful InnerSource projects, we noticed that many people began using the word InnerSource in a simplistic, degenerate manner. Probably the most damaging misunderstanding was that InnerSource meant outsourcing work from a busy team to another that presumably had more capacity. In general, it's easy to fall into the fallacy of thinking that effective processes are just about following certain procedures or using certain tools, without regard for the culture that makes success possible.

Discussing the problems caused by this vocabulary issue became a bonding moment for all of us at the InnerSource Commons. Many members of the Commons want to focus on the larger problems of culture change, and the distorted definitions of Innersource were emblematic of the problems they are fighting. InnerSource goes much farther than simple processes or tools, and sometimes that makes definitions more difficult to communicate in an enterprise environment.

The vocabulary issue might sound minor, but we've found again and again that it is vital, especially when introducing change, that terms are clearly and explicitly defined and agreed upon. It is also important that the definitions are easy to find. This includes terms like "InnerSource," but also includes roles and responsibilities. Some of the first steps toward InnerSource are to clearly and publicly define standards, roles, and responsibilities.

We Have GitHub Enterprise, So We Must Be InnerSource!

NOTE

TL;DR

GitHub helps with code transparency, but doesn't actually change the typical enterprise silo-based mentality. Without the processes we talk about in this book, GitHub instead creates serious collaboration and integration issues, which can turn into bottlenecks, especially on critical codebases needed by many teams.

The idea that GitHub is all that's needed to be InnerSource is a concept we fight against daily. Most people do not realize that it takes much more than GitHub to find, create, and grow open source communities. The communities create the software, not the other way around, but more often than not, large companies lack a sense of holistic community.

InnerSource Is About Culture and Processes, Not Just Tools

The first steps toward InnerSource must be to foster trust and increase clear communication. This makes it possible for a sense of community to grow and improves collaboration. But businesses often lead from the *how*, rather than the *why*. We can't tell them to foster trust in their teams; that is too vague and can't be expressed as an action item. We fight the constant battle of process and hierarchy versus agility and customer influence. So how do we work around that? How do we make better decisions and collaborate more, without spending more money? These are things that GitHub cannot answer but InnerSource can.

It is true that using a tool like GitHub to make version control easy, visible, and accessible is a step in the right direction. But we need to think beyond tools and their advantages and flaws, and consider people. Enterprises are made of people with their own fears, habits, established patterns, hierarchy, and motivations, and they respond to corporate politics as much as to technology. This is why each of the checklist items focuses on some aspect of the human piece of the puzzle.

A Parable: GitHub Without InnerSource

The first big problem we encountered when introducing InnerSource was an increase in escalation up the management chain. We like to call it the "Big Cheese Story" (see the following sidebar). At its core, it is a story of fear. We believe it is unique to the corporate environment and not the open source world. We found that this story resonated with many of our participants in the InnerSource Commons. The awesome part is that open source's existing processes already had several pieces of the solution, though they had not been put together before.

The Big Cheese Story

Once upon a time, there was a company that decided to embrace InnerSource, so it dictated that all code was to be moved to GitHub Enterprise. Because there was no cohesive version control before, there was much rejoicing across the company. Now, the developers finally had visibility to one another's code! No longer would the developers need to submit a change request to planning, and hope it was accepted and scheduled some time in the next year. Visions of seamless collaborations danced in developers' heads!

An intrepid programmer decided to do a pull request on the big codebase, and told her manager that she could write the necessary change in a matter of weeks. Her Big Cheese was very pleased. So, the intrepid programmer wrote the change and submitted the pull requests and waited...and waited...and waited...until her Big Cheese became very unhappy and asked why the changes had not been added. The intrepid programmer replied that she had finished the work and submitted the pull request, but the changes hadn't been accepted by the other codebase.

So, her Big Cheese went to the Big Cheese that owned the other codebase, and asked him to force one of his groups to accept those changes. After all, there was now a big backlog waiting to go through! The Big Cheese of the codebase agreed and ordered some poor individual in his group to accept those changes. But that individual didn't like how the intrepid programmer wrote the changes, because they were not "how things are done." They were written in a different style, used a different test scheme, and maybe didn't take advantage of an existing module. Thus, the second programmer rewrote the entire change before adding it to the codebase.

No one learned anything. No documentation was created. And now everyone hates InnerSource because it creates bottlenecks and makes programmers look difficult to the Big Cheeses. Plus, the product owners are frustrated because no one has included them in the process.

Breaking Down the Big Cheese Problem

At first, we were surprised by the Big Cheese problem, because we knew that the more code that other teams can see into, the better they can understand the pieces they are integrating with and/or reusing. But we found that just having the code visible doesn't automatically lead to collaboration, especially in an enterprise environment. The incentives are different from the open source environment.

First, most of the code had previously been developed in silos. This meant that teams had undocumented styles, structures, and practices that outsiders couldn't know about until they submitted code that could not be accepted by the maintainers. Beyond that, there was a siloed culture that encouraged people to talk only to people in their own group and to use language that outsiders couldn't understand. This often happens in complex structures. I'll cover the organizational aspects later in this booklet.

If you do not understand the underlying architecture of the full stack, especially an older one with poor documentation, making silos is a normal response. It is easier to understand and take ownership of only your component. This is the piece you have the most control over. However, this method doesn't lend itself to an atmosphere of collaboration, and can increase complexity and discourage reuse.

Because of this complexity, potential collaborators didn't want to contribute until the pain from lack of integration was more painful than the fear of contributing. And the owners of the codebase were loath to accept responsibility for code that was not their priority and was written by someone not on their team. This resistance to collaboration resulted in a constant stream of escalations up the leadership chain. It turned GitHub into a bottleneck, especially for high-demand codebases, which tend to be high risk. Consequently, the code that the most people needed access to became the most difficult to contribute to.

We found that contributors were often inspired to write pull requests for the changes they needed in other codebases. But the codebase hosts were not accepting their pull requests, mainly because it meant extra work and responsibility for them. It became all risk and little gain.

We had to go back in the history of open source to find answers to these cultural problems. Then, we had to figure out how to make the solutions match enterprise structures. And we had to simplify the solutions so that they could be more universally adopted. I'll explain our solution in Chapter 3, *The Most Important Role, and the First Step: Trusted Committer*.

More Communication Pitfalls

Communication in a siloed culture presents problems that are very different from those in a traditional open source environment. In particular, planning is significantly different. Two weeks before the beginning of my employment at PayPal, several teams submitted their feature-level integration requests to one popular codebase as a part of their quarterly planning. The core team took those stories and rewrote them to fit the current construct of their codebase, with no involvement of the submitters, and then sent them back to the external teams. Near the end of the quarter, many team leads came to the InnerSource team with the rewritten stories, complaining that InnerSource was really just "conscripted code." They felt like they were being conscripted to do another team's work. They did not realize that the code they were being asked to produce was actually the changes needed to complete their own integration requests. Confused, we sent them back their original requests and showed how the new stories were actually derived from their original stories. This is when we learned that the external teams had not been

involved at all in the rewrite. Clearly, this was a major communication failure! For the next (and all subsequent) rounds of planning, we made sure all teams were present for the negotiations and rewrites of stories. After this communication problem was solved, we made significant gains. An order of magnitude of code was accepted through pull requests, and external stories that had been on backlogs for years were cleared.

We also had an issue with teams creating significant pull requests against codebases with little to no warning to the codebase owners. Of course, in an enterprise environment, such a significant expenditure of resources without planning or communication too often became a battle of the Big Cheeses.

We needed to create a defined list of proven practices based on our experiences. We could see that better communication and well-defined expectations across teams was absolutely necessary. Chapter 6, *Working Within the Enterprise: Understanding Planning* presents an explanation of the steps we've taken toward a solution.

CHAPTER 3

The Most Important Role, and the First Step: Trusted Committer

NOTE

TL;DR

- For projects with any level of risk, you need to have a *Trusted Committer*. Define the role's responsibilities clearly, based on the level of risk.
- Trusted Committers shift back and forth between coding and Trusted Committer responsibilities.
- The Trusted Committer role is difficult, and you need to reward those employees who deserve and accept the role.
- The rewards to the enterprise are great: better integrated code, better code reviews, faster pull request (PR) turnaround time, clearer knowledge for refactoring, more documentation with less pain, and bottleneck reduction.

In the previous chapter, we described some of the cultural problems we've encountered. Codebase owners must accept pull requests, or they create bottlenecks and escalations up the management chain. External teams must learn and conform to the style and standards of the codebase to which they are contributing, or their contributions must be extensively rewritten. And when codebase owners and external contributors don't work together, nothing gets better and everyone ends up discouraged.

Many of the problems stem from the fact that developers in the enterprise environment are often unwilling to dedicate time to reviewing and accepting pull requests or mentoring developers in other areas. And who can blame them? They typically have assigned tasks and goals that are specific to their own project, not to other projects that happen to touch their codebase. In addition, most people are disinclined to accept responsibility for something they have not written.

But, for InnerSource to work, some developers *must* take on responsibilities outside of their silos, so we created a new role with defined responsibilities and called it the Trusted Committer (TC). This is the most fundamental change we have implemented so far, and it is crucial to making InnerSource work. In fact, it is step one in its implementation.

Defining the Role

The TC has the following list of responsibilities (each bullet point helps the TC's team to better communicate and collaborate with other teams):

- Write and maintain the rules for contributing to the codebase
- Review incoming code (pull requests)
- Mentor contributors from other areas
- Merge pull requests
- Take the lead on refactoring and modularization
- Participate in discussion lists
- Send announcements
- Watch for and suggest opportunities for collaboration

We should point out that in the open source world, many groups have independently evolved a similar role. We specifically borrowed from "The Apache Way", a tool developed by the Apache Software Foundation. These roles are assigned to people who have shown a high level of dedication to a project. TCs are ultimately responsible for the codebase, and are often gatekeepers of the code. The level of power in these roles often has a direct relationship to the amount of risk. For example, the Linux kernel is widespread and high risk, so the Linux kernel has divided its version of the TC role into two lev-

els, Janitors and Mentors. On the other hand, Node.js modules are very low risk. The community might not embrace a new module after vetting it, but new modules can't break anything, so there is no TC role. Anyone can publish a Node.js module with npm.

Refining the Role

After we had a defined role for the TC, we found a new problem: developers didn't like the role, because they were afraid of getting too far away from the code; they didn't want to lose coding time. They also struggled with prioritizing between coding and TC tasks. Plus, it was costly in time and attention for them to switch too frequently between those tasks. It made it difficult to get into the coding zone. To solve this issue, we considered removing programmers from active coding and assigning them the TC role as a full-time job. But this came with its own problem: we agreed that when people stop contributing code themselves, it becomes increasingly difficult for them to review code, especially integrations. Not to mention that it made programmers depressed because they would no longer be doing the kind of work they loved and entered the field to do.

Our solution is to have the TCs work with the product specialists to create a rotation schedule for themselves. They publish their schedules for other teams to see, in order to manage contributor expectations. We also find it helpful to list each TC's specialties in the schedule so that the contributors know when someone with the appropriate expertise will be available to help them. It was also important to create new reward structures for the difficult and critical work done by TCs, a process I'll describe later in the section "Rewarding TCs."

In our experience, the number of TCs per project varies greatly. In a high-risk project with about 30 developers, we ask that six programmers be assigned to the TC role. At any one time, half of them actively work in the TC role, reviewing code and mentoring, while the other half actively code. They switch roles at the end of every two-week sprint. This has been ideal for the TCs, because two weeks is a good solid length of time to either really get into coding, or to settle into mentoring and documentation.

In our lower-risk projects like tooling, a single TC works on 10 repos or more. Most developers are very eager to mentor contributors on their toolsets. This is ideal for helping teams across enterpri-

ses figure out how to better standardize their toolsets because everyone is welcome to contribute. The main suggestion we have for those TCs is to have office hours so that they can maintain blocks of time to get (and stay) in the coding zone.

Immediate Benefits

Assigning the code reviews of PRs[1] to the TC role greatly accelerated the turnaround on the PRs and increased the level of code reviews. Plus, we found that TCs used their mentoring time to create some wonderful documentation for the next big refactor of code. The lead for one of the major architectural reworks said that using InnerSource helped his team really understand how to significantly refactor the codebase. It also greatly decreased the amount of interrupt-driven coding from external bug fixes because those were also addressed in the bug fix PRs.

The documentation was created semi-painlessly by archiving public mentorship discussions between the TCs and contributors, and making them easily accessible in a context-relevant location in the codebase itself. This meant that the time spent on mentoring, valuable in and of itself, served double duty. We call this passive documentation, and we discuss it in more depth in Chapter 4, *Passive Documentation and the Need for Findability*.

Rewarding TCs

We found it important to work with HR and management to ensure the TC role is recognized formally. This solves two problems: development wins because they are reassured that management must respect the code review process, and no more Big Cheeses forcing code changes! The enterprise wins because the new role gives a path to promote programmers without taking them away from coding, which is what they do best and often love the most.

The TC role illuminates a developer's advanced skills in mentoring, deep knowledge of architecture, and best code-review practices. We have found the TC role to be a difficult one, and companies need to

1 GitHub uses the term PR, as do several other tools. Companies not using these tools might call the same thing problem reports, change requests, or tickets.

determine how to properly reward those dedicated staff that take on the additional responsibilities.

We are enhancing our promotion path to Fellow for developers to reflect this complexity. This allows us to reward the "full-stack" developers we are creating and allows promotion without having to move to management roles that some developers find to be tedious. We get to keep the programmers that really understand the various codebases and encourage them to help refactor and reduce technical debt.

CHAPTER 4

Passive Documentation and the Need for Findability

NOTE

TL;DR

- Passive documentation is crucial for mentoring and capturing tribal knowledge. The team takes a communication hit at the beginning, but the increase in velocity more than makes up for it.
- You can accelerate passive documentation by rewarding both the writers and consumers of the document.
- Passive documentation must be findable to be useable. Sometimes, this means that you will need to manually cross-tag between siloed datasets.

Passive documentation is the record of the documentation we create every day while communicating openly. It is a great way to get tribal knowledge out of silos and into a format that is archival and findable. As an added bonus, it is typically kept with the project or the code that it documents, thus it is in an easy-to-find, context-relevant location.

Creating Passive Documentation

Passive documentation consists of written information that was produced not specifically to document for the future, but to explain

something in the present, as it is needed. For example, it often includes the following:

- Conversations that the Trusted Committers (TCs) have while mentoring a contributor who is learning how to integrate with her codebase
- Conversations the product owners have when they are explaining their priorities to one another, or arranging them
- The connection between a piece of the code and the project stories about the code, and the live conversations about both

At first, the most difficult part is persuading people to have these conversations more openly. They tend to start out wary of creating a lasting reference document on the fly. We found that when people realize that they are not writing formal documents, but are simply capturing mentoring conversations, the resistance dissipates. And the benefits of the rapid increase in documentation are quickly obvious to all.

To be captured in passive documentation, conversations need to happen in a written format. Common written formats include comments in a pull request, a tagged conversation in a public Slack channel, a comments page in a wiki, and a tagged email in a discussion group. In the open source world, we often say that conversations that don't happen publicly on the email list or wiki aren't "real." We are working to change the culture internally to be the same. If there is an important discussion in person, at the end of it one person always commits to creating a written record of it. They do this by writing the discussion up in an email that all parties can approve, and then posting the write-up to the larger community.

"Did You Read the FINE Manual?"

We found that after the TCs had answered a few easy questions publicly on pull requests, the velocity of the next contributor's pull request immediately increased.

Diligent contributors search the documentation before asking for help, or even writing their pull requests. In our case, we store this in GitHub, and because everything is in GitHub, there is little ambiguity about where to look. We encourage the TCs to refer contributors

back to previous conversations when they do not incorporate previous advice in their pull requests.

We are working on ways to reward these public conversations internally. We are creating dashboards that highlight when someone has written especially relevant documentation. And we allow TCs to reward contributors who do their research first. Trust me, the TCs will quickly learn who follows directions and will prioritize their pull requests first!

Findability

In the open source world, when you want to find out how to do something, you simply Google it. In the corporate world, finding information is much more difficult. Most information is locked away in different software and datastores that might or might not be searchable. Often the information in these applications is locked down by default, because that seems safer. But in the long run it is very damaging to a company. Locking information away makes onboarding a new employee a difficult process, and it makes integrating a new acquisition almost impossible. Moreover, it invites, or even encourages, an atmosphere of tribal knowledge.

Sometimes, those difficulties are created by the tools themselves when they have a bad or nonexistent search function. Sometimes, there are just so many tools being used that aggregation becomes an issue. Too often, problems are aggravated by pricing issues that force the company to shell out additional fees to enable access for all users.

But documentation is only useful if people can find it, so this is a really important problem to solve. Many of our teams have begun requiring cross-tagging spanning application silos in order to enable manual searching. For example, we have had several teams decide in their contributing agreement that they will not even consider a pull request that does not have a searchable tag of some sort, for example, a JIRA number for a bug fix, or a Rally story number for a feature-level pull request in GitHub. This is a huge help when someone needs to manually search across multiple locked-up datastores, but it isn't ideal, and it requires developers to be quite diligent.

We have begun creating tools to assist in finding and sharing information. We created (and open sourced!) RallySlack. When someone

is on Slack, RallySlack automatically pulls up all of that individual's Rally stories to make it easier to find and tag a Slack conversation. With RallySlack, users don't need to look up or memorize Rally story numbers. We are developing a similar tool for GitHub to help with tagging Rally story numbers in pull requests and issues. Eventually we hope to open source this tool, as well.

CHAPTER 5

Creating Good House Rules for Guests: Writing Contributing Agreements

NOTE

TL;DR

- Trusted Committers (TCs) are responsible for writing contributing agreements to explain house rules to contributors (e.g., code conventions and dependencies). Contributing agreements are living documents.
- Contributors need to be good houseguests and read the agreements (and any other findable documentation) before contributing. The better they groom their contribution to match the contributing agreement, the greater the velocity of acceptance.
- Management needs to support the TCs on these agreements.
- Be careful when standardizing agreements because this leads to less ownership by the TCs. Complex agreements can prevent contributions and should be reserved for high-risk projects.

TCs cannot be forced to accept and take ownership of broken code, code without proper tests, undocumented code, or even code that

doesn't meet their style standards. Contributing agreements are a way to formalize the responsibilities of the developers on the originating side of the code.

What Is a Contributing Agreement?

The TCs write and own their *contributing agreements*. A contributing agreement is a device that specifies the house rules to let contributors know what is required in order for the TC to accept a code contribution. Contributing agreements are viewable by everyone in development. They must have the TCs' names, contact information, and schedule. After that, the content is up to the TC. It will likely include some of the following:

- The authoring TC's specialties
- Community guidelines
- Code conventions
- Testing conventions
- Branching conventions
- Commit-message conventions
- Steps for creating good pull requests
- How to submit feature requests
- How to submit bug reports
- How to submit security issue reports
- How to write documentation
- Definition of done
- Dependencies
- Build-process schedule
- Sprint schedule
- Road map

It is very important for the TCs to be able to invoke these agreements for protection. If another team's code contribution does not meet the receiving TC's specifications, the TC needs to be able to point to the contributing agreement to explain exactly why the code is being rejected. This helps immensely to minimize corporate politics and escalation issues.

Mi Casa Es Su Casa

The contributing agreements are also crucial in managing a contributor's expectations. The metaphor we use is that of house rules for guests. Everything goes more smoothly if hosts communicate their expectations to their guests, instead of assuming that everyone has the same standards. Someone with a nice house with many breakable things and a very organized kitchen will have different house rules from a person who lives in a comfortable mess with cat-scratched furniture.

And hosts should warn guests about quirks in their house, like a circuit breaker that trips if someone tries to run the microwave and the dishwasher at the same time. The contributing agreement is the perfect place to list the house rules and pitfalls of your codebase. And, like clearly explained house rules, it can prevent damage, misunderstandings, and hurt feelings.

The metaphor extends to contributor behavior. Good guests follow the house rules, of course, but they also tidy up; that is, they help fix bugs or refactor code. And a *great* guest brings a bottle of wine! A great codebase guest might contribute a feature or fix that everyone likes and wants.

Win/Win

Most contributors quickly realize that the more closely their submissions adhere to the contributing agreements, the faster those submissions are accepted and committed. Also, contributors know that when they see a more permissive agreement, there is less risk in submitting changes.

One Size Fits All?

In the open source world, different groups have different rules for contributions. Most of the differences are risk related. The Linux kernel has very strict submittal guidelines and processes that go far beyond a simple contributing agreement. On the other hand, agreements for Node.js modules are very permissive; they mostly ask that people do a search to ensure that they aren't duplicating someone else's effort.

This diversity is very similar to the variety of projects in an enterprise. We all have certain core projects that could topple the business if they fail, and these projects require strict contributing agreements. But we also have tools that we would like to standardize, and this is a much lower-risk activity. The toolset teams should have the flexibility to have simpler contributing agreements to lure people into collaborating. Often, these less-risky codebases can be safe places for contributors to learn how to participate in InnerSource projects.

Creating the contributing agreements is a balance between safety and participation. A short, easy agreement indicates that you welcome contributions and are willing to mentor people through the process of contributing. A longer, more complex agreement can convey difficulty, risk, and the fact that contributors need to pass several goals before their code will be accepted.

Some groups have tried to standardize one contributing agreement across the entire company. This is a pretty natural reflex for large enterprises. But we have fought against this because a company-wide agreement takes ownership away from the TCs, costing the company their buy-in, and eliminates the flexibility just outlined. Instead, we create templates as a starting place for TCs (such as the list in "What Is a Contributing Agreement?" on page 26), adjusted for various levels of risk and complexity. We also ask that TCs revisit and update their contributing agreements after a retrospective or when new TCs are assigned to the codebase. It is vital that contributing agreements remain living documents.

CHAPTER 6

Working Within the Enterprise: Understanding Planning

NOTE

TL;DR

- Transparency needs to be a part of the planning process. Creating internal transparency has led in our experience to more than an order of magnitude gain in external code acceptance.
- Create formal processes to work within the enterprise environment. Formalizing processes keeps everyone on the same page.
- Transparency in planning helps because if the employees do not understand why decisions are made, they cannot propose corrections to the implementation. Top-down management is a complex process that rarely works. Open collaboration scales better.

The biggest difference between InnerSource and open source is the business structure and its constraints. Working within an enterprise means a constant pull of hierarchy and power structures that are often contrary to the basic ethos of transparency and individual agency that is key to open source. Yet, open source has much to offer the business world. So how do we adapt to the business environment without diluting the fundamental aspects of open source?

Keep It Small and Simple, and Engage Your Staff

Our biggest successes have resulted from finding and using a key point of leverage within the existing structures in the enterprise. We review current processes and find places to modify them in small ways to move incrementally toward InnerSource. We work with the business environment's desire to work with *how*s and not *why*s, and simply tell them explicitly how to modify processes to improve outcomes, without going into lectures about transparency and ownership. It is best to make the changes as simple as possible, both to encourage adaptability and to avoid triggering the resistance large organizations can have to change.

For example, our written process for creating contributor agreements is very small and simple with few requirements: the agreements are owned by the Trusted Committers (TCs), they are viewable by other teams, and they contain the TCs' contact information and availability. Other than the contact information and schedule, we do not dictate the content of the agreements at all. Of course, we do encourage and expect them to contain much more information! And problems with a guest contributor become the ideal learning experience to trigger additions or changes to the contributing agreement. It's kind of like when you stay at someone's house, and the host has a rule of no loud music after 2 a.m.; you know that someone before you must have played loud music at 2 a.m.

A crucial part of the TC process is that the employee who will be most affected by the change is given more power (and more responsibility) to manage that change.

These simple requirements, plus the rule that TCs are completely in charge of accepting or rejecting code changes, are relatively small and unalarming changes to the InnerSource process. But look at the results:

- The TCs have a huge incentive to fully participate in the new process.
- Better communication and documentation begins as soon as the agreement goes beyond contact information.
- The explicit expectations laid out in the document lead to better collaboration.

- Code changes move more quickly, leading to a positive feedback loop.
- As more code changes come in, the TCs do more mentoring, which creates more documentation.
- The TCs become more deeply familiar with their codebase and its external impact.

The minimal requirements allow the teams to adapt the process to their own needs—a major tenet of open source—and lead to the InnerSource goals of better collaboration, fewer bottlenecks, better integration, and, almost certainly, cleaner code.

Planning and Product Specialists

After our success in improving integration with TCs and contributor agreements, we knew we had to create something similar to smooth the planning process. The product specialist role, which monitors all aspects of the product lifecycle, needs to work on breaking down silos between teams and products, and to see how these products can integrate with others in the company.

Product specialists need the ability and knowledge to properly negotiate and prioritize features across teams. But, we have found that even though people working on code or product integration *know* they need to sit down and discuss things with the other teams involved, they don't usually make time for the necessary meetings unless they're pushed. Anything not on the schedule is easy to put off. This results in poor communication, delays, and misunderstandings. The fix is a formal process change to force the necessary meetings, with greater inclusion to ensure that the appropriate people are in the planning sessions, and greater transparency to break down the silo mentality. We are working with our product specialists now to improve public records of this process.

Inclusion and Transparency

Full inclusion in the planning stage of the process is crucial; all of the teams must be at the table for the process to work smoothly. Representatives from each team need to be present for the story grooming process, not just the owners of the primary codebase. Different teams often have different or conflicting priorities. Getting

planners from each team together in one room helps them negotiate among themselves to get all the work done. Inclusion leads to smoother collaboration.

Transparency during the planning process is also important. We feel that it helps to reduce conflict. When a conversation is public and intended to be archived, we find that participants often become better at considering the entire company when working out priorities. It helps to break down the silo mentality.

Transparency in planning increased as a side effect of adding more people to the planning meetings. More people are present for the trade-offs and negotiations. Just as important is the small process change we had already implemented, requiring that all relevant conversations be a part of the passive documentation. This means that everyone can review discussions in the future, and alters people's conversational strategies. Also, by creating passive documentation, you can avoid information overload as people search more and spam less.

Prioritization of projects and resources is usually done opaquely at companies. The reasoning is rarely made public and is done behind closed doors. This leaves employees to come up with their own narratives to explain priorities. Again, we see that when a company gives the *how* but not the *why*, employees cannot make adjustments on the fly. It cripples their decision-making, and is a key element of bad escalation processes.

Bringing transparency to the process gives employees the ability to make corrections as necessary, because they understand the end goal and will not blindly continue down a designated path that they know will lead to the wrong outcome. In addition, making prioritization and resource allocation more transparent reduces hierarchically based fears of kingdom building, or the appearance of it.

Planners Can Have an Impact on Processes

We began with simple rules. For high-risk and high-demand codebases, we found it necessary to formalize planning. The product specialists worked with the TCs to add rules to the contributing agreements requiring external contributors to file an issue request before submitting a significant code change. "Significant" meant using more than three story points in Rally. This requirement was

the first step in creating a solidified process that requires the product specialists on both teams to meet and collaborate with one another as well as the TC and contributor prior to a significant code change.

Such structured meetings were not necessary for other codebases. Instead, their contributing agreements generally ask potential contributors to contact the TC and product specialist in advance on the listed discussion channels. Again, adaptability is key!

Results

Greater inclusion in the planning stages does create a resource problem initially: scheduling meetings with large groups is difficult, the meetings can run longer than anticipated, and every person pulled into a meeting necessarily is putting off other tasks. But we saw benefits almost immediately. The teams understood the prioritization process better, which improved our Agile process. And the change velocity in the core team's ability to accept external code was so large that it more than made up for the time lost in planning, by a factor of 10. And we were able to clear stories from contributor's teams that had been on the backlog for years.

Opening up the process by including more people and making it more transparent also has an amazing effect on the teams' ability to cross-collaborate. This leads to more effective decision-making both internally and across teams. We also found that by improving communication through passive documentation, eventually the meetings became smaller as teams used clearer communication.

We did find that the increased communication required some external facilitation in the beginning. A key element was teaching the product specialists to negotiate more effectively by always looking at the win/win solution for the company. This stage was relatively short; after things were ironed out between teams, collaboration increased dramatically and little external help was needed.

Crossing the Gap from Planning to Developers

To our great satisfaction, the teams at PayPal really began to work well together, doing some horse trading and some very complex bargaining. One team's product specialist even came up with a new, simple process change that we have added to the InnerSource check-

list: product specialists must create and own a file called *HELPWANTED.md*. This file is where product specialists can transparently post their backlog. Developers who are looking for a project to work on will search the code repositories, but don't usually think to look into project management tools, even if they have access. So, the *HELPWANTED.md* file is placed in the code repository. Again, findability is important!

Some well-done *HELPWANTED.md* documents have been generated from the project management tools, complete with notes and levels of prioritization. This really helps inform guest contributors about other teams' needs. Often, potential contributors can tell which stories are similar to their own projects, so they choose which ones they can help with the most. The *HELPWANTED.md* file is a great addition to the win/win mentality. Contributors can and do trade with other teams, fixing an item in the backlog in exchange for moving their feature-sized code changes up the review list.

CHAPTER 7

From Internal Silos to Internal Transparency

NOTE

TL;DR

- Team divisions tend to create silos of information that hamper decision-making.
- Transparency can be reintroduced through rules about documenting activities and requirements for meetings on important decisions.

Where Did Silos Come From?

British anthropologist Robin Dunbar has proposed a theory that a social animal's neocortex size limits that animal's comfortable social group size. Dunbar's number suggests that humans are best able to handle approximately 150 relationships in a larger sense, and tend to maintain stronger ties to smaller groups of around 50.[1] This maps back to tribes and their sizes. Many in the corporate world have noticed similar size limitations when creating organizational charts. Silos are a natural extension of those hierarchical situations. They also evolve out of specialization, and from these groupings tribal knowledge is born.

1 Ipsita Priyadarshini, "All you want to know about Dunbar's number", visionteme-nos.com.

With silos, people do not need to care as much about consensus. They make communication and permissions easier because they divide people into smaller groups. And dividing people into assigned groups can make accountability easier. If something goes wrong with security, people can blame the security group.

What's Wrong with Silos?

As you can guess, communication between the silos can become difficult, and problems can cascade through the company. Communicating over a wall never has the same emotional impact on people. Getting things right matters more when you know one of the customer representatives personally and you hear his frustration when a bug you accidently created increases his call volume.

In the modern world of integrations and acquisitions, silos seriously hamper cross-team communications. Silos make the world of permissions a devastating morass of unproductivity as people swim upstream to find the tribal knowledge locked away in the silos. In really large enterprises, people attempt to diagram the convoluted jungle paths of interactivity and permissions via road maps. This is a losing battle because the vines continue to grow even as maps are drawn.

Transparency for Community Sourcing

Hopefully, by now you are convinced that transparency in coding has multiple benefits. But consider this, as well: transparency is a core tenet in science and academia. It is a key element in scientific rigor.[2] We believe that it works within a company for the same reasons it works with academia. It improves collaboration, and empowers anyone within the company to see when a project is going off track and help to fix it.

Many businesses are learning the value of transparent communication with customers. Facebook, Twitter, and other forms of social media empower customers to get issues resolved publicly. Busi-

2 *https://en.wikipedia.org/wiki/Transparency_(behavior)*

nesses have the opportunity to increase trust by handling problems effectively.[3]

Transparency Boosts Decision-Making

Many organizations are seriously hampered because people can't offer input to decisions that cross team boundaries. A staff member will say, "I don't know what's really going on in that group, so I can't make a judgment." The documentation and open meetings that this report proposes as part of the InnerSource culture will go a long way toward fixing this problem. People need three things to participate in decision-making: information (which documentation provides), access (which comes in meetings), and permission (which must become part of the InnerSource culture).

How Do We Break Down Silo Walls?

If the problem with silos is that they impair communication and lock up tribal knowledge, the solution is to create processes that open communication channels and produce findable documentation.

Luckily, we can do both with the same processes, discussed in earlier chapters. All of the steps toward InnerSource are steps toward opening up the silos:

- Create new roles to take ownership of creating and maintaining documents that explicitly communicate expectations and needs, or assign this responsibility to existing roles. The roles should go to people most directly affected by the changes.
- Create processes that require inclusive, in-person meetings during the planning stages of integrations that are openly documented.
- Require discussions and announcements to use designated public channels.
- Make those channels accessible company-wide.
- Archive the discussions in a findable location.

3 *https://www.theguardian.com/sustainable-business/rebuild-restore-recover-trust-business*

There is a cost to making these changes. Projects might slow down as some people move into new roles and others adjust to more meetings and online discussions. But the benefits come quickly.

Findable Documentation Is Part of Transparency

We as a society have become accustomed to using a search to find what we need. Filing email into folders is becoming obsolete; why spend time filing when you can just run a search? Yet finding necessary documentation in a company can feel more like a treasure hunt. One of our big goals is to improve documentation, and a major component of this is not just *creating* it, but making it *findable*. Yes, documentation is an aspect of transparency.

Documentation in accessible and logical places becomes a major driver of transparency and collaboration. Creating documentation is usually a low priority and rarely has passionate champions, yet it can drastically shorten learning curves, ease collaboration, and prevent misunderstandings. Fortunately, InnerSource, when done correctly, creates extensive and findable documentation as a side effect.

As an enterprise transitions toward InnerSource, there is a time cost to the extra communication that is required. But if the enterprise first sets up passive documentation processes to capture this extra communication, that extra communication is a huge gain for future productivity. Learning from what worked and what didn't is always useful for organizations, but most enterprises make no real effort to capture and share this knowledge. Sometimes, it is fear of liability, but if the conversations are done in a public fashion, people will already be aware of those repercussions and higher-quality conversations will ensue.

Look for ways to make passive documentation available to all levels in the company, not just developers in GitHub, email lists, or Slack conversations. Passive documentation reduces the barrier to nonwriters sharing knowledge precisely when it is needed. And by having the immediate feedback that passive documentation creates, questions are guaranteed to be answered well. Slack and other tools that allow more communication across silos have been extremely valuable in increasing collaboration.

Of course, there's probably already some existing documentation scattered across the company. Many of our member organizations at the InnerSource Commons are looking at larger search solutions to make it easier to open information across departments and tools. Although their solutions don't let people change or edit what they find in search (they must to return to the tool that created the information), they can at least find out what is happening and to whom to talk. Slack is proving very valuable in both facilitating and archiving those discussions on a wider scope.

It is also important for the enterprise to make collaboration a real priority. Communication costs will temporarily rise as the organization transitions. This is typical for in any change-management scenario. But communicating earlier during cross-team collaboration creates large productivity gains. Companies spend millions and millions on internal integrations and integrations of acquisitions. Having these conversations publicly facilitates the next acquisition or integration.

Where Do We Still Need to Improve Transparency?

Those of us doing InnerSource already have code transparency through GitHub. We have taken the first steps to make planning and communications more transparent, but we still need to find more solutions in this area.

Often, tools themselves inhibit transparency across departments. When most enterprise software charges by the user, it becomes financially prohibitive to buy a user account for everyone in the company. A solution is to look primarily at tools that allow the company to become tools-agnostic through APIs. Many tools today have APIs, so you can use tools like Zapier and IFTTT to connect them.

What Are the Limits or Pitfalls in Enterprise Transparency?

There are hard limits to transparency in commercial enterprises, especially for publicly traded companies and international companies that need to worry about compliance issues with multiple governments. This is a significant difference from most open source

organizations. Another important pitfall is handling remote access. Again, this is largely because of regulatory issues. When looking for technological solutions, you must keep these issues in mind.

Some enterprise agencies do look to open source organizations to help with global transparency when they are ready to become universally open. But when wading through the difficulties of being open within the company, while not violating laws about insider trading, they are on their own.

Still, the push to recover lost tribal knowledge, combined with the increased productivity and employee morale engendered by InnerSource, is convincing many enterprises that the price of transparency, even with all of these caveats, is a worthwhile endeavor.

There is also the topic of information overload and overcommunication. This is why search is a key element. We want to transition corporate culture from a push mentality, in which endless bulletins are sent out, to a pull mentality, in which people are confident that they will get the information they need when they need it via search.

CHAPTER 8

Looking Forward

During our journey, we have found a need for many tools. Some help facilitate discussion and some help with standardization and compliance; others help with measurement and reporting. Please join us at InnerSourceCommons.org where we are working on the open source versions of these tools.

One such tool is called Agora—for enterprise search. We are working toward an open system in which employees can easily add in diverse data sources. This will allow search across tools and domains.

We also are discussing maturity levels at the Commons. The first pass has been in regard to GitHub and GitLab metrics. But we would like to measure reuse and collaboration across data sources. However we can do this only if we first capture the data.

Creating an Industry Standard

We have created an organization called InnerSource Commons. Currently, we have more than 50 members, most from enterprise-sized organizations. One of our primary goals at the moment is to create an industry standard. We are working on creating pattern languages from stories that our members create.

We are spreading information in several ways:

- We are working with O'Reilly Media to create books (like this one) and training materials to help teach other people and their companies about InnerSource.
- We have classes based on ones we've given at conferences, now trimmed to fit in 30-minute segments.
- We have training materials on the wiki. If you have any feedback or create any materials that you want to share, please contact us there or follow the link to our Slack chat channel.

InnerSource Pattern Language

One very large-scale project under way at the Commons is creating a pattern language for finding solutions to problems. Leonardo da Vinci looked to nature for solutions to difficult problems. When *we* encounter a difficult problem, we look to an open source collection of previously solved problems that have a pattern similar to ours. In the pattern project, we create simple patterns that contain five elements:

- A description of the problem
- The larger context around the problem
- The forces that must be considered in finding a solution
- A possible solution
- The new context that results from applying the solution

Thankfully, the many similar (and already documented!) patterns in the open source world are making quick work of this project.

CHAPTER 9
Appendix

The Actual Checklist

It's easy to feel overwhelmed with the task of starting an InnerSource project, much less taking your organization through the transition to a true InnerSource or open source company. PayPal has drawn a checklist from its own experiences and the experiences of colleagues in other companies making the transition. We present it here to help organize and focus what you need to do. Certainly, following the checklist slavishly will not produce success. Every organization needs to undertake investigations, discussions, and cultural change in order to benefit from InnerSource. But keeping this checklist in front of you can help reassure you that you're making progress.

This report was inspired by both the GNU Manifesto and Checklist Manifesto. We hope to inspire you to take at least a small step toward InnerSource and creating your own checklist and sharing it with us. The Apache Way was an important inspiration for the team, and you will spot similarities in this list.

- Readiness
  - Personal
    - Do you believe this is a viable strategy for your company, or are you just doing it out of ideological commitment and idealism?

    - Do you understand the changes required to be successful at InnerSource?
    - Are you skillful at presenting ideas to others, listening to their concerns, and finding common ground without bias?
- Project
    - Does this project matter to the company? Is it likely to survive strategy changes?
    - Appropriateness
        - Is this project likely to be interesting to developers outside the original development team?
            - Is it currently or could it be used widely by other teams in the company who depend on it?
            - Will it benefit from being extended by outsiders in ways that the original team could not anticipate?
        - Could the project benefit from having other teams contribute bug fixes and refactoring work?
        - Could outside developers contribute useful code or suggestions?
        - Would outside developers respond to appeals to contribute?
    - Code maturity
        - Is the project modular enough to make changes easy and safe to make?
        - Is the code well documented?
    - Existing process
        - Can releases be made frequently?
        - Is continuous testing and integration in place?
        - Is all of the code stored in a version control repository such as GitHub that makes branches, pull requests, and integration easy?
- Team
    - Are team members ready for the challenges of InnerSource?

    - Accepting code and changes to their code from outsiders
    - Being responsible for less-than-perfect code contributed by outsiders
    - Having outsiders see their less-than-perfect code
    - Having to conduct conversations that are sometimes difficult with outsiders about accepting and rejecting their contributions
    - Mentoring and/or learning to mentor contributors
- Do team members understand the requirements of running an InnerSource project? (It helps if members have participated on open source projects.)
    - Creating and maintaining documentation for guest contributors
    - Willingness to participate in forums and answer questions patiently
    - Using forums, which provide a historical record everyone can see, instead of hallway conversations (optionally, using a scribe to make notes during live meetings, just as long as all of the decisions and explanations for those decisions are documented)
    - Willingness to do code reviews of outside submissions
    - Maintaining the bug tracker
    - Taking a turn in the role of TC
- Have you chosen TCs?
    - Do they understand their responsibilities?
        - Write and maintain the *CONTRIBUTING.md* file in GitHub
        - Review incoming code (pull requests)
        - Mentor guest contributors
        - Merge pull requests
        - Take the lead on refactoring and modularization
        - Participate and answer questions on discussion lists

        - Send announcements
        - Watch for and suggest opportunities for collaboration
    - Do they understand the potential rewards (intrinsic and job-related) of this position?
        - Develop deeper understanding of the codebase
        - Improve the quality of your team's code
        - Improve the quality of code within the organization as a whole with better integrations
        - Learn and grow as a developer by seeing many incoming code examples
        - Understand how to better refactor and modularize code to encourage external contributions
        - Develop interpersonal and leadership skills through mentoring and negotiation with guest contributors
    - Have they been given leeway to do things not previously in their job descriptions?
- Is time set aside in team members' work schedules for these new responsibilities?
- Are team members trained to handle these responsibilities?
- Is there a mechanism for making announcements that anyone in the organization can follow and search? (Examples: Slack, email)
- Is there a recorded mechanism for discussion so that all Guest Contributor questions and internal team decisions are searchable by incoming Guest Contributors? (Examples: Slack, online forum)

- Company
    - CxO-level executives
        - Do the executives understand the purpose and value of running a project as InnerSource?
            - The value of spreading tribal knowledge across the organization

        - The value of having teams remove their own external blockers and bottlenecks
        - The value of a more interconnected organization
        - The value of having advanced team members that have a deeper understanding of many arms of the codebase
        - How InnerSource encourages basic good practice and helps develop and spread even better coding and development practices
        - How InnerSource increases the learning and development of individual developers
    - Are they willing to support flexible work requirements and time spent on cross-departmental contributions?
    - Can they accept experimentation, failure, and repositioning?
    - Are they willing to support a path for career advancement that does not require management?
    - Does the executive team support the initiative?
    - Are company goals and KPIs clearly stated and shared?
    - Are the company's vision and mission relevant, up-to-date, clear, respected, and followed?
- Human resources
    - Are there rewards and criteria for promotions based on InnerSource values?
        - Rewards for people who contribute to projects in other departments
        - Rewards for developers who handle contributions from other departments (TCs)
    - Need path for advancement that respects community role and does not require moving into management
- Organizational setup
    - Central coordinating team
        - Is the team set up?

        - Is it ready to communicate with projects interested in going InnerSource? Can it apply the criteria in this checklist to ensure that the project and team are ready?
        - How do projects register as InnerSource?
    - Do staff members have time to contribute to outside projects?
    - Do staff members have resources to measure and demonstrate gains and losses of teams?
    - Can staff members choose what projects to work on based on their expertise and motivations?
    - Is there a meritocratic philosophy that can appreciate good contributions from all corners?
- Developers
    - Do developers around the company understand that they can contribute to the InnerSource project?
    - Do they understand the value of contributing to other projects?
        - Removing external blockers
        - Building integrations with other tools themselves
        - Seeing how other teams structure code and learn from examples
    - Do they understand the process for making contributions to other projects?
        - Joining the discussion
        - Reading the contribution requirements
        - Exploring and/or contributing to the Help Wanted file
        - Signing up for announcements
    - Do they know how to use the tools?
        - Version control
        - Programming language
        - Test development
    - Do they know how to support their team's role in InnerSource?

        - Can they participate productively on forums?
            - Answer contributor questions
            - Describe the reasons for choices that have been made in a way that is clear
            - Respond constructively to feedback
        - Can they participate in dialogs around reviews of their code?
    - Are they permitted to contribute to projects outside their departments?
        - Do they understand how to get support from their product owners and managers?

- Repository
    - Are ancillary resources set up, such as the following?
        - Documentation
        - Discussion forum
        - Bug tracker
        - Wiki
    - Are the following files in place?
        - *README.md*
            - Project name
            - Any earlier names and codenames for this project
            - Project description
            - Team lead and PM/PMO contact information
            - Keywords for search purposes
                - So that people can find this particular project by name
                - So that people can find this project by what it does
            - How to sign up for the announcement list
            - Location of discussion forum
            - Location of other repositories (Examples: JIRA, Rally, Confluence)
        - *CONTRIBUTING.md*

    - Required
        - Table of contents
        - Names and contact information for TCs
        - TC availability schedule
    - Optional
        - Community guidelines
        - Code conventions
        - Testing conventions
        - Branching conventions
        - Commit-message conventions
        - Steps for creating good pull requests
        - How to submit feature requests
        - How to submit bug reports
        - How to submit security issue reports
        - How to write documentation
        - Dependencies
        - Build process schedule
        - Sprint schedule
        - Road map
        - Helpful links, information, and documentation
        - When the repositories will be closed to contributions
- *HELPWANTED.md*
    - Can be initially empty
    - Can link to a forum where requests and offers are posted
    - Can contain a list of requests and offers
- *GETTINGSTARTED.md*
    - Whatever a contributor might need to get the app up and running to begin coding
    - Can be filled in by an intern, or after some contributors get started, based on feedback about what would have been helpful to them to get started

    - Who reviews the documents in the repository?
- Tools
    - Version control
    - Continuous integration
    - Testing
- Logistics
    - Sprints
    - Codeathons, especially for tools
    - Education
    - Testing
    - Gamification
    - Reporting
        - Pull requests during recent period
            - Number
            - Size (in lines of code)
            - Type
            - Name of pull request submitter
        - Baseline metrics
        - Ongoing tracking
            - Resources
                - Costs
            - Time
        - Who sees measurements? Display to the community?
    - Bug fixing
- Product owners

About the Author

Silona Bonewald has been a member of the open source community since the late '90s, and is currently director of InnerSource at Paypal.